**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчёт по лабораторной работе №2

«Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных.»

Выполнил: Проверил:

студент группы РТ5-61Б преподаватель каф. ИУ5

Агеев А Гапанюк Ю.Е.

Подпись и дата: Подпись и дата:

Москва, 2023 г

# Описание задания

* Выбрать набор данных (датасет).
* Для первой лабораторной работы рекомендуется использовать датасет без пропусков в данных, например из [Scikit-learn.](https://scikit-learn.org/stable/datasets/toy_dataset.html)

Для лабораторных работ не рекомендуется выбирать датасеты большого размера.

* Создать ноутбук, который содержит следующие разделы:

1. Текстовое описание выбранного Вами набора данных.
2. Основные характеристики датасета.
3. Визуальное исследование датасета.
4. Информация о корреляции признаков.

* Сформировать отчет и разместить его в своем репозитории на github.

Дополнительно примеры решения задач, содержащие визуализацию, можно посмотреть в репозитории курса mlcourse.ai - <https://github.com/Yorko/mlcourse.ai/wiki/Individual-projects-and-tutorials-(in-Russian)>

# Ход работы

import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline   
sns.set(style="ticks")

# Будем использовать только обучающую выборку  
data = pd.read\_csv('data/lab2\_prepared\_2.csv', sep=",")

# размер набора данных  
data.shape

(234, 12)

# типы колонок  
data.dtypes

Country or Dependency object  
Population(2020) object  
Yearly Change float64  
Net Change object  
Density(p/km^2) float64  
Land Area(km^2) object  
Migrants float64  
Fert Rate float64  
Med Age float64  
Urban float64  
World Share float64  
Regions object  
dtype: object

# проверим есть ли пропущенные значения  
data.isnull().sum()

Country or Dependency 0  
Population(2020) 0  
Yearly Change 0  
Net Change 0  
Density(p/km^2) 0  
Land Area(km^2) 0  
Migrants 33  
Fert Rate 33  
Med Age 33  
Urban 13  
World Share 0  
Regions 11  
dtype: int64

# Первые 5 строк датасета  
data.head()

Country or Dependency Population(2020) Yearly Change Net Change \  
0 China 1.439.323.776 0.39 5.540.090   
1 India 1.380.004.385 0.99 13.586.631   
2 Indonesia 273.523.615 1.07 2.898.047   
3 Pakistan 220.892.340 2.00 4.327.022   
4 Bangladesh 164.689.383 1.01 1.643.222   
  
 Density(p/km^2) Land Area(km^2) Migrants Fert Rate Med Age Urban \  
0 153.000 9.388.211 -348.399 1.7 38.0 61.0   
1 464.000 2.973.190 -532.687 2.2 28.0 35.0   
2 151.000 1.811.570 -98.955 2.3 30.0 56.0   
3 287.000 770.880 -233.379 3.6 23.0 35.0   
4 1.265 130.170 -369.501 2.1 28.0 39.0   
  
 World Share Regions   
0 18.47 NaN   
1 17.70 Asia   
2 3.51 Asia   
3 2.83 Asia   
4 2.11 Asia

total\_count = data.shape[0]  
print('Всего строк: {}'.format(total\_count))

Всего строк: 234

# Обработка пропусков в данных

# Удаление колонок, содержащих пустые значения  
data\_new\_1 = data.dropna(axis=1, how='any')  
(data.shape, data\_new\_1.shape)

((234, 12), (234, 7))

# Удаление строк, содержащих пустые значения  
data\_new\_2 = data.dropna(axis=0, how='any')  
(data.shape, data\_new\_2.shape)

((234, 12), (190, 12))

data.head()

Country or Dependency Population(2020) Yearly Change Net Change \  
0 China 1.439.323.776 0.39 5.540.090   
1 India 1.380.004.385 0.99 13.586.631   
2 Indonesia 273.523.615 1.07 2.898.047   
3 Pakistan 220.892.340 2.00 4.327.022   
4 Bangladesh 164.689.383 1.01 1.643.222   
  
 Density(p/km^2) Land Area(km^2) Migrants Fert Rate Med Age Urban \  
0 153.000 9.388.211 -348.399 1.7 38.0 61.0   
1 464.000 2.973.190 -532.687 2.2 28.0 35.0   
2 151.000 1.811.570 -98.955 2.3 30.0 56.0   
3 287.000 770.880 -233.379 3.6 23.0 35.0   
4 1.265 130.170 -369.501 2.1 28.0 39.0   
  
 World Share Regions   
0 18.47 NaN   
1 17.70 Asia   
2 3.51 Asia   
3 2.83 Asia   
4 2.11 Asia

# Заполнение всех пропущенных значений нулями  
# В данном случае это некорректно, так как нулями заполняются в том числе категориальные колонки  
data\_new\_3 = data.fillna(0)  
data\_new\_3.head()

Country or Dependency Population(2020) Yearly Change Net Change \  
0 China 1.439.323.776 0.39 5.540.090   
1 India 1.380.004.385 0.99 13.586.631   
2 Indonesia 273.523.615 1.07 2.898.047   
3 Pakistan 220.892.340 2.00 4.327.022   
4 Bangladesh 164.689.383 1.01 1.643.222   
  
 Density(p/km^2) Land Area(km^2) Migrants Fert Rate Med Age Urban \  
0 153.000 9.388.211 -348.399 1.7 38.0 61.0   
1 464.000 2.973.190 -532.687 2.2 28.0 35.0   
2 151.000 1.811.570 -98.955 2.3 30.0 56.0   
3 287.000 770.880 -233.379 3.6 23.0 35.0   
4 1.265 130.170 -369.501 2.1 28.0 39.0   
  
 World Share Regions   
0 18.47 0   
1 17.70 Asia   
2 3.51 Asia   
3 2.83 Asia   
4 2.11 Asia

### Обработка пропусков в числовых данных

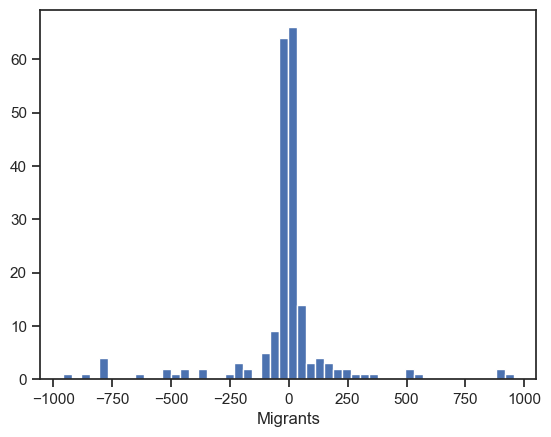
# Выберем числовые колонки с пропущенными значениями  
# Цикл по колонкам датасета  
num\_cols = []  
for col in data.columns:  
 # Количество пустых значений   
 temp\_null\_count = data[data[col].isnull()].shape[0]  
 dt = str(data[col].dtype)  
 if temp\_null\_count>0 and (dt=='float64' or dt=='int64'):  
 num\_cols.append(col)  
 temp\_perc = round((temp\_null\_count / total\_count) \* 100.0, 2)  
 print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp\_null\_count, temp\_perc))

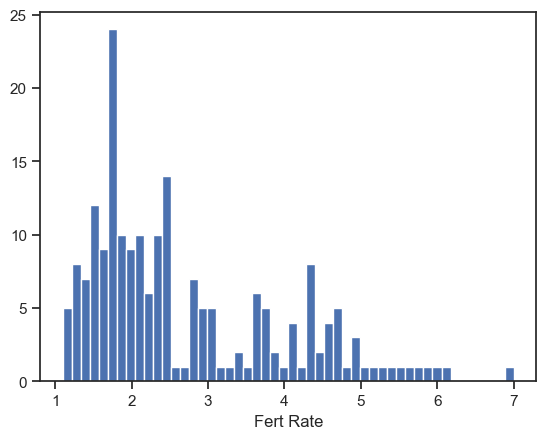
Колонка Migrants. Тип данных float64. Количество пустых значений 33, 14.1%.  
Колонка Fert Rate. Тип данных float64. Количество пустых значений 33, 14.1%.  
Колонка Med Age. Тип данных float64. Количество пустых значений 33, 14.1%.  
Колонка Urban. Тип данных float64. Количество пустых значений 13, 5.56%.

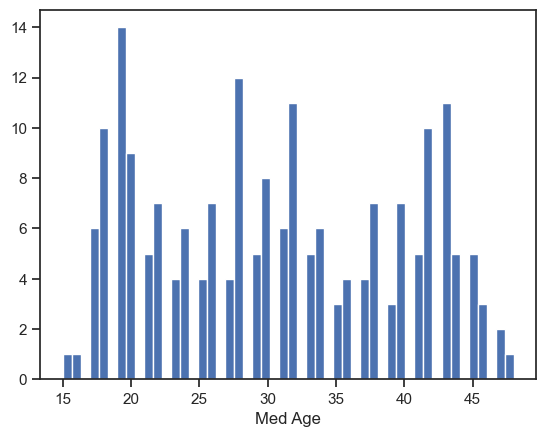
# Фильтр по колонкам с пропущенными значениями  
data\_num = data[num\_cols]  
data\_num

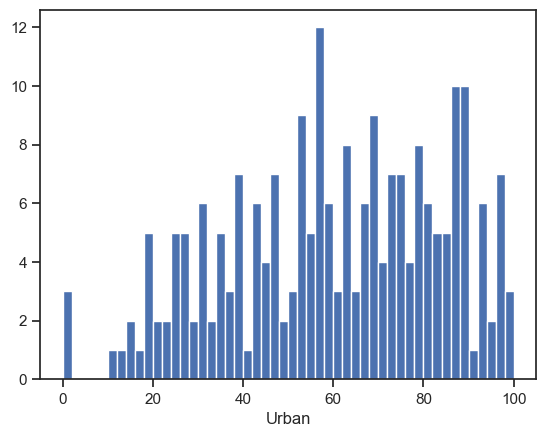
Migrants Fert Rate Med Age Urban  
0 -348.399 1.7 38.0 61.0  
1 -532.687 2.2 28.0 35.0  
2 -98.955 2.3 30.0 56.0  
3 -233.379 3.6 23.0 35.0  
4 -369.501 2.1 28.0 39.0  
.. ... ... ... ...  
229 NaN NaN NaN 62.0  
230 NaN NaN NaN 0.0  
231 NaN NaN NaN NaN  
232 NaN NaN NaN 46.0  
233 NaN NaN NaN 0.0  
  
[234 rows x 4 columns]

# Гистограмма по признакам  
for col in data\_num:  
 plt.hist(data[col], 50)  
 plt.xlabel(col)  
 plt.show()









data\_num\_Fert\_Rate = data\_num[['Fert Rate']]  
data\_num\_Fert\_Rate.head()

Fert Rate  
0 1.7  
1 2.2  
2 2.3  
3 3.6  
4 2.1

from sklearn.impute import SimpleImputer  
from sklearn.impute import MissingIndicator

# Фильтр для проверки заполнения пустых значений  
indicator = MissingIndicator()  
mask\_missing\_values\_only = indicator.fit\_transform(data\_num\_Fert\_Rate)  
mask\_missing\_values\_only

strategies=['mean', 'median', 'most\_frequent']

def test\_num\_impute(strategy\_param):  
 imp\_num = SimpleImputer(strategy=strategy\_param)  
 data\_num\_imp = imp\_num.fit\_transform(data\_num\_Fert\_Rate)  
 return data\_num\_imp[mask\_missing\_values\_only]

strategies[0], test\_num\_impute(strategies[0])

('mean',  
 array([2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398,  
 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398,  
 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398,  
 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398,  
 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398, 2.6920398,  
 2.6920398, 2.6920398, 2.6920398]))

strategies[1], test\_num\_impute(strategies[1])

('median',  
 array([2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3,  
 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3,  
 2.3, 2.3, 2.3, 2.3, 2.3, 2.3, 2.3]))

strategies[2], test\_num\_impute(strategies[2])

C:\Users\prite\anaconda3\lib\site-packages\sklearn\impute\\_base.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.  
 mode = stats.mode(array)

('most\_frequent',  
 array([1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8,  
 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8,  
 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8]))

# Более сложная функция, которая позволяет задавать колонку и вид импьютации  
def test\_num\_impute\_col(dataset, column, strategy\_param):  
 temp\_data = dataset[[column]]  
   
 indicator = MissingIndicator()  
 mask\_missing\_values\_only = indicator.fit\_transform(temp\_data)  
   
 imp\_num = SimpleImputer(strategy=strategy\_param)  
 data\_num\_imp = imp\_num.fit\_transform(temp\_data)  
   
 filled\_data = data\_num\_imp[mask\_missing\_values\_only]  
   
 return column, strategy\_param, filled\_data.size, filled\_data[0], filled\_data[filled\_data.size-1]

data[['Urban']].describe()

Urban  
count 221.000000  
mean 59.714932  
std 23.818160  
min 0.000000  
25% 43.000000  
50% 62.000000  
75% 79.000000  
max 100.000000

test\_num\_impute\_col(data, 'Urban', strategies[0])

('Urban', 'mean', 13, 59.71493212669683, 59.71493212669683)

test\_num\_impute\_col(data, 'Urban', strategies[1])

('Urban', 'median', 13, 62.0, 62.0)

test\_num\_impute\_col(data, 'Urban', strategies[2])

C:\Users\prite\anaconda3\lib\site-packages\sklearn\impute\\_base.py:49: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.  
 mode = stats.mode(array)

('Urban', 'most\_frequent', 13, 57.0, 57.0)

### Обработка пропусков в категориальных данных

# Выберем категориальные колонки с пропущенными значениями  
# Цикл по колонкам датасета  
cat\_cols = []  
for col in data.columns:  
 # Количество пустых значений   
 temp\_null\_count = data[data[col].isnull()].shape[0]  
 dt = str(data[col].dtype)  
 if temp\_null\_count>0 and (dt=='object'):  
 cat\_cols.append(col)  
 temp\_perc = round((temp\_null\_count / total\_count) \* 100.0, 2)  
 print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp\_null\_count, temp\_perc))

Колонка Regions. Тип данных object. Количество пустых значений 11, 4.7%.

cat\_temp\_data = data[['Regions']]  
cat\_temp\_data.head()

Regions  
0 NaN  
1 Asia  
2 Asia  
3 Asia  
4 Asia

cat\_temp\_data['Regions'].unique()

array([nan, 'Asia', 'Africa', 'Europe', 'Latin America & Caribbean',  
 'Northern America', 'Oceania'], dtype=object)

cat\_temp\_data[cat\_temp\_data['Regions'].isnull()].shape

(11, 1)

# Импьютация наиболее частыми значениями  
imp2 = SimpleImputer(missing\_values=np.nan, strategy='most\_frequent')  
data\_imp2 = imp2.fit\_transform(cat\_temp\_data)

# Пустые значения отсутствуют  
np.unique(data\_imp2)

array(['Africa', 'Asia', 'Europe', 'Latin America & Caribbean',  
 'Northern America', 'Oceania'], dtype=object)

# Импьютация константой  
imp3 = SimpleImputer(missing\_values=np.nan, strategy='constant', fill\_value='NA')  
data\_imp3 = imp3.fit\_transform(cat\_temp\_data)

np.unique(data\_imp3)

array(['Africa', 'Asia', 'Europe', 'Latin America & Caribbean', 'NA',  
 'Northern America', 'Oceania'], dtype=object)

data\_imp3[data\_imp3=='NA'].size

11

# Преобразование категориальных признаков в числовые

cat\_enc = pd.DataFrame({'c1':data\_imp2.T[0]})  
cat\_enc

c1  
0 Africa  
1 Asia  
2 Asia  
3 Asia  
4 Asia  
.. ...  
229 Oceania  
230 Africa  
231 Oceania  
232 Oceania  
233 Africa  
  
[234 rows x 1 columns]

## Кодирование категорий целочисленными значениями (label encoding)

from sklearn.preprocessing import LabelEncoder

cat\_enc['c1'].unique()

array(['Africa', 'Asia', 'Europe', 'Latin America & Caribbean',  
 'Northern America', 'Oceania'], dtype=object)

le = LabelEncoder()  
cat\_enc\_le = le.fit\_transform(cat\_enc['c1'])

# Наименования категорий в соответствии с порядковыми номерами  
  
# Свойство называется classes, потому что предполагается что мы решаем   
# задачу классификации и каждое значение категории соответствует   
# какому-либо классу целевого признака  
  
le.classes\_

array(['Africa', 'Asia', 'Europe', 'Latin America & Caribbean',  
 'Northern America', 'Oceania'], dtype=object)

cat\_enc\_le

array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,  
 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2,  
 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,  
 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,  
 3, 3, 3, 3, 3, 3, 0, 3, 0, 4, 0, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5,  
 5, 5, 5, 5, 5, 5, 5, 0, 5, 5, 0, 5, 5, 0])

np.unique(cat\_enc\_le)

array([0, 1, 2, 3, 4, 5])

# В этом примере видно, что перед кодированием   
# уникальные значения признака сортируются в лексикографиеском порядке  
le.inverse\_transform([0, 1, 2, 3])

array(['Africa', 'Asia', 'Europe', 'Latin America & Caribbean'],  
 dtype=object)

## Кодирование категорий наборами бинарных значений

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()  
cat\_enc\_ohe = ohe.fit\_transform(cat\_enc[['c1']])

cat\_enc.shape

(234, 1)

cat\_enc\_ohe.shape

(234, 6)

cat\_enc\_ohe

<234x6 sparse matrix of type '<class 'numpy.float64'>'  
 with 234 stored elements in Compressed Sparse Row format>

cat\_enc\_ohe.todense()[0:10]

matrix([[1., 0., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.],  
 [0., 1., 0., 0., 0., 0.]])

cat\_enc.head(10)

c1  
0 Africa  
1 Asia  
2 Asia  
3 Asia  
4 Asia  
5 Asia  
6 Asia  
7 Asia  
8 Asia  
9 Asia

pd.get\_dummies(cat\_enc).head()

c1\_Africa c1\_Asia c1\_Europe c1\_Latin America & Caribbean \  
0 1 0 0 0   
1 0 1 0 0   
2 0 1 0 0   
3 0 1 0 0   
4 0 1 0 0   
  
 c1\_Northern America c1\_Oceania   
0 0 0   
1 0 0   
2 0 0   
3 0 0   
4 0 0

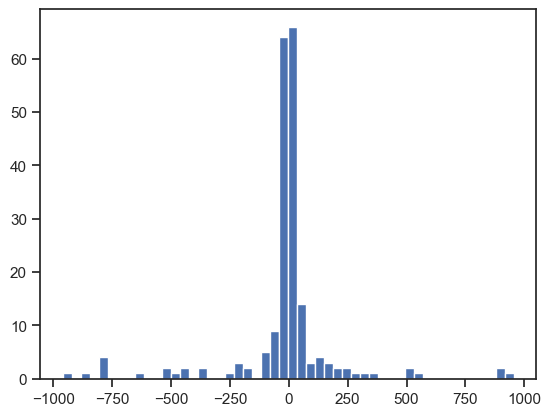
pd.get\_dummies(cat\_temp\_data, dummy\_na=True).head()

Regions\_Africa Regions\_Asia Regions\_Europe \  
0 0 0 0   
1 0 1 0   
2 0 1 0   
3 0 1 0   
4 0 1 0   
  
 Regions\_Latin America & Caribbean Regions\_Northern America \  
0 0 0   
1 0 0   
2 0 0   
3 0 0   
4 0 0   
  
 Regions\_Oceania Regions\_nan   
0 0 1   
1 0 0   
2 0 0   
3 0 0   
4 0 0

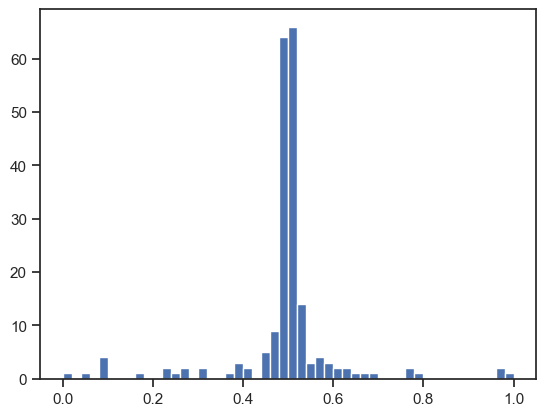
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer

sc1 = MinMaxScaler()  
sc1\_data = sc1.fit\_transform(data[['Migrants']])

plt.hist(data['Migrants'], 50)  
plt.show()



plt.hist(sc1\_data, 50)  
plt.show()



## Масштабирование данных на основе Z-оценки

sc2 = StandardScaler()  
sc2\_data = sc2.fit\_transform(data[['Migrants']])

plt.hist(sc2\_data, 50)  
plt.show()

